Markus Mäkelä

Klusteroidun SQL-tietokannan suorituskykyyn vaikuttavat tekijät

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

24.11.2014

Metropolia

Tekijä Otsikko Sivumäärä Aika	Markus Mäkelä Klusteroidun SQL-tietokannan suorituskykyyn vaikuttavat tekijät 33 sivua 28.1.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Vesa Ollikainen VP Engineering Rasmus Johansson

Pilvipalveluiden yleistyminen ja suuri suosio yritysmaailmassa asettavat alati kasvavia vaatimuksia näiden palveluiden perustana toimiville klusteroiduille tietokannoille. Näiden tietokantaklustereiden suorituskyky on täten oleellinen kysymys niin yrityksille kuin näiden palveluiden käyttäjille.

Tämän insinöörityn tarkoituksena oli selvittää, mitä tarkoitetaan klusteroidun tietokannan suorituskyvyllä, mitkä tekijät vaikuttavat siihen ja miten sitä voidaan parantaa. Työ aloitettiin jakamalla suorituskyvyn käsite kahteen osaan: suoritetun kyselyn vasteaikaan ja klusterin kykyyn suorittaa yksittäinen kysely. Näiden kahden määritelmän avulla määriteltiin tekijät, joiden perusteella klusterin suorituskykyä voidaan parantaa.

Näistä tärkeimmäksi nousi klusterin varsinainen topologia, eli miten yksittäisen tietokantapalvelimet on sijoitettu ja minkälaisen suhteen ne muodostavat toisiinsa, ja miten replikaatio oli toteutettu tietokantaklusterissa. Näiden tekijöiden perusteella tutkittiin keinoja, joilla voitaisiin parantaa klusteroidun tietokannan suorituskykyä. Teoriassa suorituskyvyllä ei ole kattoa, mutta käytännössä saatavilla olevat resurssit ovat aina suurin rajoittava tekijä, joka vaikuttaa tietokantaklusterin suorituskykyyn. Tämän perusteella yleiset ratkaisut eivät välttämättä ole paras tapa ratkaista tietokannan suorituskykyongelmia.

Tämän päätelmän pohjalta tätä insinöörityötä varten toteutettiin palomuurimoduuli MaxScale-tietokantavälityspalvelinta varten. Tämän moduulin avulla ratkaistiin kolme esimerkkitapausta, joiden kohdalla ongelma oli tarkasti määritelty ja täten ne olivat helposti ratkaistavissa. Moduulin testauksen ja toimivuuden varmistamisen jälkeen pohdittiin suorituskykymittausten hyödyllisyyttä ja sitä, kuinka ne tulisi tehdä. Pohdinnan lopputulos oli suorituskykymittausten tarpeellisuuden tunnistaminen ja tosiasia, että käytettävien ja hyödyllisten mittaustuloksien tuottaminen on työlästä ja tarkkuutta vaativaa työtä.

Avainsanat	SQL, klusterointi

Author(s) Title Number of Pages Date	Markus Mäkelä Factors Affecting the Performance of Clustered SQL-databases 33 pages 28 January 2015
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Specialisation option	Software engineering
Instructor(s)	Vesa Ollikainen, Senior Lecturer Rasmus Johansson, VP Engineering

The growing popularity of cloud-based solution has caused a rise in demand for clustered database solutions. This growing demand is placing stricter requirements for the clustered databases and so the performance of these databases is a vital question for the corporations providing these services as well as the users of these services. The goal of this thesis was to define what performance means for a clustered database performing some sorts of replication, what factors affect it and how to improve it. First the concept of performance was divided into two parts where the first part was the actual time it took to finish a single query and the second part was how available the database was i.e. how likely it is that the query will succeed. After this definition of performance the key factors that affect it were identified. The most critical factor that could be controlled was found to be the way data replication was handled inside the database cluster and how the nodes in the cluster were placed globally. Finally, after identifying the key factors for performance some general performance improvements could be made and the most reasonable in most cases would be to increase the number of nodes in the replication cluster. In the end, customized solutions would always beat general optimization procedures and due to this fact a module for the MaxScale database proxy was created as a part of this thesis.

The module that was created as a part of this thesis was a simple firewall that blocks SQL-queries based on a set of rules. Three example cases were used when the module was designed, created and tested. After the verifying that the module indeed produced a performance increase some thought was given to how usable are performance tests.

The main dilemma was that highly tuned performance tests might not actually test the test subject and might give a too positive picture of the outcome. In the end a conclusion was reached which states that performance tests are generally needed but great care is required when designing them.

Keywords	SQL, cluster

Sisällysluettelo

Lyhenteet

SQL	Structured Query Language, relaatiotietokantojen hallintaan sunniteltu kieli

1 Johdanto

Tämän insinöörityön aiheena on klusteroitujen tietokantojen suorituskyky ja miten se ilmenee käytännössä. Insinöörityön motivaationa oli viimeisen vuosikymmenen aikana tapahtunut tietotekniikan kehitys. Uudet tavat käyttää tietoa ja tiedon kasvu, niin määrältään kuin laadultaan, ovat luoneet kasvavan tarpeen korkean virhetoleranssin järjestelmille, jotka pystyvät palvelemaan suuria käyttäjämääriä. Esimerkkejä näistä järjestelmistä ovat sosiaalisen median sivustot, kuten Facebook ja Twitter. Myös yritysmaailma on siirtymässä kohti käytäntöä, missä tieto on tallennettu johonkin korkeasti saavutettavissa olevaan verkkopohjaiseen järjestelmään. Tiedon ja toiminnallisuuden keskittäminen yhteen järjestelmään ei kuitenkaan ole konseptina täysin uusi. Suurtietokoneet olivat ensimmäisiä keskitettyjä järjestelmiä, johon yksittäiset käyttäjät ottivat yhteyden. Näiden suosio yritysmaailmassa laski kuitenkin jyrkästi 1990-luvun alussa, kun henkilökohtaiset tietokoneet nousivat suosioon [1; 2.]

Tämän insinöörityön kautta saadaan selville, mitä on klusterointi relaatiotietokantojen tapauksessa, kuinka sitä käytetään, miten oleellista sen suorituskyky on ja kuinka sitä parannetaan. Työ antaa myös työkalut suorituskyvyn syvempään analyysiin ja antaa käytännön esimerkin, kuinka suorituskykyongelmia voi ratkaista.

Ensimmäiseksi käsitellään työn tarkoitus ja tavoitteet, joiden pohjalta työtä on tehty. Tämän jälkeen kuvataan insinöörityön aiheen toimintaympäristö, joka luo perustan seuraavalle osiolle, jossa käsitellään klusteroidun tietokannan suorituskykyä. Tämä osio pitää sisällään suorituskyvyn määritelmän, jonka pohjalta eritellään suorituskykyyn vaikuttavia tekijöitä. Tähän erittelyyn perustuen tunnistetaan erilaisia keinoja, joilla suorituskykyä voidaan parantaa ja millaisiin tilanteisiin ne sopivat.

Suorituskyvyn vaikutuksista tehdään käytännön tutkimus, jonka perustana on tämän opinnäytetyön osana kehitetty palomuurisuodatin SQL-välityspalvelimeen. Tätä moduulia, ja sen kehitystä, kuvataan kolmen esimerkkitapauksen kautta. Käytännön osuus aloitetaan laatimalla moduulin vaatimukset ja tavoitteet, minkäj jälkeen kuvataan sen toteutus ja perustelut toteutetuille ominaisuuksille. Moduulin testauksen ja toimivuuden varmistamisen jälkeen pohditaan sen soveltuvuutta testiongelmien ratkaisuksi. Lopuksi tehdään yhteenveto klusteroitujen tietokantojen suorituskyvystä ja siitä, kuinka se ilmenee palomuurimoduulin käytössä.

Tämän työn tavoitteina on selvittää, mitä tarkoitetaan klusteroidun tietokannan suorituskyvyllä, mitkä tekijät vaikuttavat siihen ja millä keinoilla sitä voidaan parantaa. Työn aihe on tärkeä, koska nykyinen IT-teknologia on suuntaamassa kohti mobiilia pilviympäristöä, jossa suurin osa sovelluksista ja palveluista joko sijaitsee verkossa tai on tekemisissä sen kanssa. Suurin osa näistä tarvitsee jonkinmuotoista tallennustilaa tai on vuorovaikutuksessa jonkin toisen palvelun kanssa, ja suurin osa näistä käyttää niin sanottua pilvimallia datan tallentamiseen. Yleensä näiden perustana ovat klusteroidut SQL-tietokannat, joten niiden suorituskyky on oleellinen kysymys monelle yritykselle ja yhteisölle.

Ensimmäisenä tavoitteena on selvittää, mitä tarkalleen ottaen klusteroidun tietokannan suorituskyky on, miten sen voi määritellä ja millä tavalla se ilmenee käytännössä. Tämän tiedon avulla tutkitaan ja arvioidaan keinoja, joilla voidaan parantaa klusteroidun tietokannan suorituskykyä.

Tämä työ keskittyy kolmeen esimerkkitapaukseen, joissa jokaisessa ongelmana on klusteroidun tietokannan suorituskyky. Työn toisena tavoitteena on kehittää palomuurimoduuli, joka toimii ratkaisuna näiden esimerkkitapauksien ongelmiin ja jonka soveltuvuutta kunkin ongelman ratkaisuksi analysoidaan.

Lopputuloksena työlle on selvitys klusteroidun tietokannan suorituskyvyn määritelmästä, siihen vaikuttavista tekijöistä ja sen parantamisen keinoista ja analyysi siitä, kuinka hyvä vastaus palomuurimoduuli on esimerkkitapausten ongelmiin.

2 Toimintaympäristön kuvaus

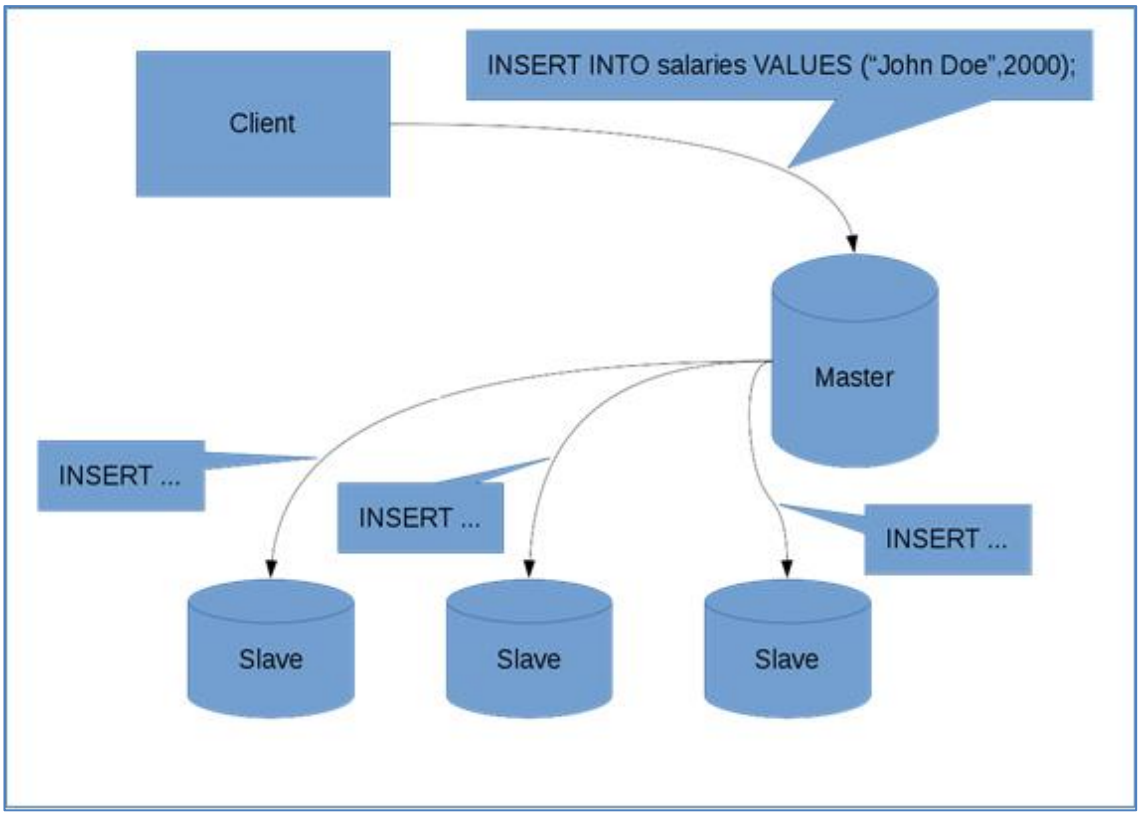
Suuri osa markkinoilla tarjolla olevista tietokannoista on relaatiotietokantoja. Vaikka nykyään on tarjolla määrällisesti paljon muitakin vaihtoehtoja perinteiselle relaatiotietokannoille, hallitsevat kuitenkin relaatiotietokannat tietokantojen maailmaa. Eräänlaisena vastalauseena tälle on muodostunut NoSQL-liike, jonka ideologiana on monimuotoisen datan säilytys tietokannoissa. Viime vuosina NoSQL:n rooli suurten tietomäärien hallinnoimisessa on kasvanut ja jatkuvasti kehittyvä tietotekniikan kulttuuri tuo entistä enemmän tapoja hyödyntää erinäisten NoSQL-ratkaisujen tuomia etuja.

Myös jatkuvasti kasvava informaation määrä on johtanut tilanteeseen, jossa tarvitaan tapoja hallita hyvin suuria määriä tietoa, ja tämän niin sanotun Big Data:n rooli suhteessa perinteisiin tietokantojen käyttötarkoituksiin on kasvamassa. Yhä useammin tämän tyyppisen datan tallentamiseen käytetään replikoituja tietokantoja [3; 4; 5.]

Tietokantojen replikaatiolla tarkoitetaan yhden tietokannan sisällön kopiointia johonkin toiseen tietokantaan. Tämän jälkeen näitä kopioituja tietokantoja päivitetään aina, kun alkuperäistä tietokantaa muutetaan. Alkuperäisiä tietokantapalvelimia kutsutaan Master-palvelimiksi ja siitä kopioituja Slave-palvelimiksi. Kyselyt, jotka eivät muokkaa tietokannan sisältöä millään tavalla, voidaan tehdä joko Master-palvelimeen tai johonkin Slave-palvelimista. Jos kysely lisää, päivittää, poistaa tai millään tavalla muokkaa tietokannan sisältöä, pitää kysely tehdä Master-palvelimessa. Tämän kyselyn suoritus, ja siitä seuraava tietokannan sisällön muutos, Master-palvelimessa käynnistää replikaatioprosessin. Master-palvelin lähettää Slave-palvelimille kopion muutoksista päivittäen niiden sisällön Master-palvelinta vastaavaksi. Tätä replikaation tyyppiä kutsutaan Master-Slave-replikaatioksi.

ACID-yhteensopivuudella tarkoitetaan sitä, kun sarja operaatioita tietokantaan voidaan suorittaa siten, että sen seuraukset ovat yksiselitteiset, eristetyt toisista operaatioista, keskeytettävissä ja että tietokanta säilyy eheänä myös näiden operaatioiden jälkeen. Operaatioista tekee yksiselitteisiä niiden täydellinen onnistuminen tai epäonnistuminen. Ne joko suoritetaan kokonaan tai ei ollenkaan. Operaatiot eivät myöskään vaikuta toisiin meneillään oleviin operaatioihin ja ovat täten toisistaan eristettyjä tapahtumia tietokannan näkökulmasta. Jos kaikkia operaatioita ei voida suorittaa tai jokin niistä epäonnistuu, voidaan kaikkien sitä ennen tehtyjen operaatioiden vaikutukset muuttaa takaisin. Kun tämä tapahtuu, tietokanta palaa operaatiosarjaa edeltävään tilaan sen tiedon kohdalla, joka operaatiosarjan käsitteli. Tietokannan eheänä säilyminen taataan sillä, että ennen kaikkia operaatioita ja kaikkien operaatioiden jälkeen tietokanta pysyy eheässä tilassa. Tietokantojen replikaation tapauksessa ACID-yhteensopivuus säilytetään, kun noudatetaan Master-Slave-peritaatetta, eli kaikki kirjoitukset tehdään Master-palvelimeen, josta ne kopioidaan Slave-palvelimiin.

Master-Slave-replikaation käytännön vaikutukset näkee kuvasta 1, jossa yhden tietueen lisääminen tietokantaan aiheuttaa muutosten leviämisen Slave-palvelimiin.



Kuva 1. Muokkaavan kyselyn replikaatio Master-Slave-klusterissa

On myös mahdollista käyttää useampaa kuin yhtä Master-tietokantaa. Tätä replikaation tyyppiä kutsutaan Multi-Master-replikaatioksi ja siinä Master-palvelimia on useampi kuin yksi. Tämä aiheuttaa datan synkronoinnissa ongelmia ja vaatii yleensä jonkin erikoistuneen ratkaisun. Tietokantapalvelimien sisältö pidetään ajantasalla erinäisiä tekniikoita käyttäen. Esimerkiksi MySQL Cluster käyttää synkroniseen datan replikaatioon kaksivaiheista transaktiota, jolla varmistetaan kaikkien tietokantasolmujen synkronisuus. Tämän avulla voidaan selvittää, mikä palvelimista pitää sisällään uusimman version tiedosta. Esimerkkejä erikoistuneista Multi-Master-tietokantaklustereista ovat MySQL Cluster, MariaDB Galera ja Galera Cluster. Nämä kaikki ovat niin sanottuja todellisia Multi-Master-toteutuksia, jotka käyttävät synkronista datan päivitystä. On myös mahdollista toteuttaa Multi-Master-replikaatio käyttämällä asynkronista datan replikaatiota. Tämä rikkoo osittain ACID-yhteensopivuuden ja mahdollistaa tilanteet, joissa tietokantasolmujen sisältö on ristiriidassa tai tiedosta on eri versioita. Tässä replikaation muodossa kaksi kyselyä, jotka muuttavat samaa tietoa, voidaan suorittaa samaan aikaan. Tästä mahdollisesti seuraa tilanne, jossa kahdessa eri Master-palvelimessa samalla tiedolla on kaksi eri arvoa. Tämän tilanteen tapahtuessa tietokannan on osattava ratkaista ristiriitatilanne jollain tavalla. Mahdollisia

ratkaisuja on esimerkiksi aikaleimaan perustuva, jossa toinen ristiriitaisista kyselyistä poistetaan aikaleiman arvon perusteella.

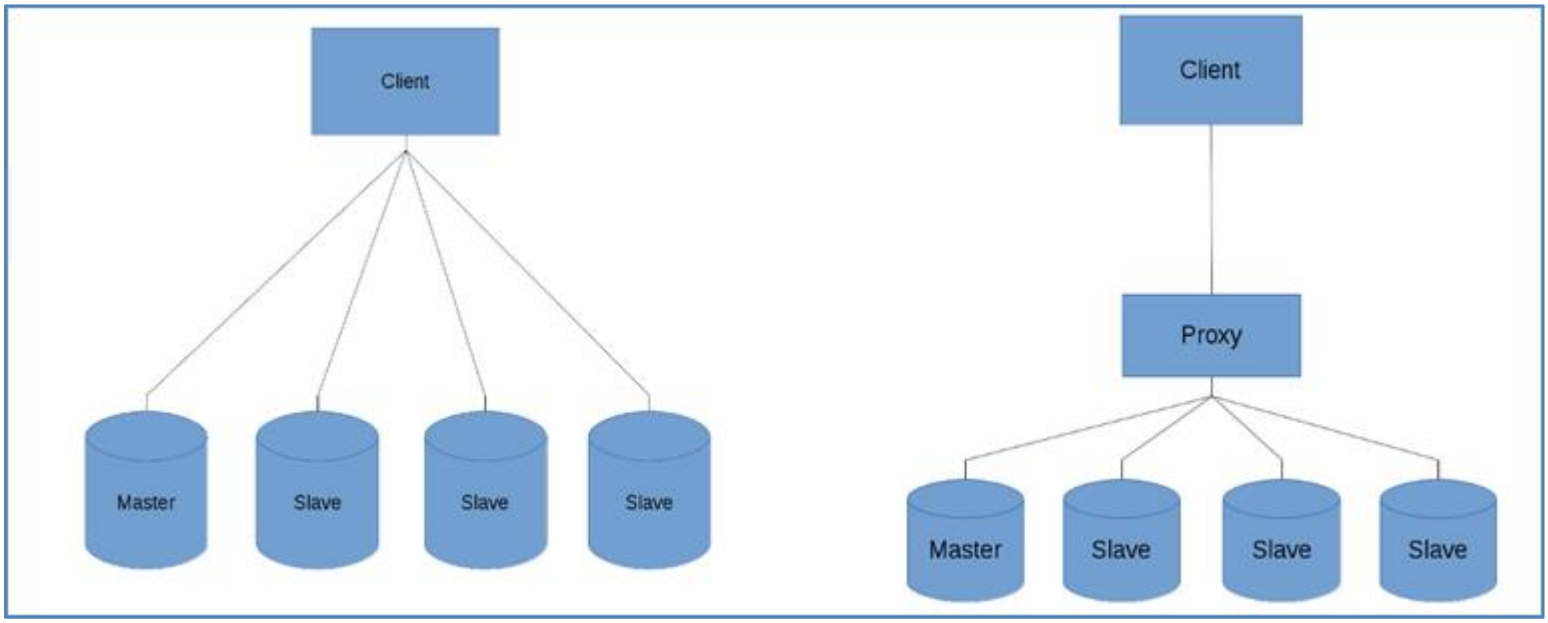
Jotta Master-tietokannan tieto olisi aina oikeassa ja ettei tietokantojen välille muodostuisi ristiriitoja, ei Slave-tietokantoihin tulisi tehdä kirjoittavia kyselyitä. Tästä ilmeneekin replikaation yleisin käyttötarkoitus, suorituskyvyn parantaminen. Tietokantaklusterin suorituskyky suuren kyselytaakan alla paranee, kun kaikki tietoa muokkaavat kyselyt kohdistetaan Master-palvelimeen ja lukukyselyt hajautetaan kaikkien Slave-palvelinten kesken. Tämä vähentää yksittäiseen palvelimeen kohdistuvaa rasitusta ja tekee tietokannan toiminnasta nopeampaa. Kyselyiden tyyppi on kuitenkin tekijänä replikaation hyödyllisyydessä. Jos kyselytaakka koostuu suurimmalta osin muokkaavista kyselyistä, joudutaan kaikki kyselyt tekemään Master-palvelimeen, ja replikaation suorituskyvylliset hyödyt jäävät minimaalisiksi. Jos kyselytaakka koostuu pääosin lukevista kyselyistä, ovat replikaation hyödyt suorituskyvyssä merkittävät. Käytännössä pelkästään Slave-palvelinten määrä rajoittaa lukukyselyiden suorituskykyä. [1] [2].

Replikaatio parantaa myös tietokantaklusterin kykyä kestää virheitä ja ongelmatilanteita. Tämä onkin replikaation toinen käyttötarkoitus: korkean saatavuuden tavoittaminen. Korkea saatavuus on oletus, että tieto on aina saatavilla ja että kyselyt onnistuvat aina. Yksittäisen Slave-palvelimen kaatuminen ei juurikaan vaikuta laajassa replikaatiota käyttävässä järjestelmässä. Edes kaikkien Slave-tietokantojen kaatuminen ei tee tietokantaklusterista toimintakyvytöntä ja vasta Master-tietokannan kaatuminen estää uuden tiedon kirjoittamisen tehden koko klusterista toimintakyvyttömän.

Myös tiedon varmistaminen on helpompaa replikaation avulla, koska vakavasti vioittuneen palvelimen tietokannan sisältö on helposti saatavilla muista tietokannoista, ja tämä tieto on automaattisesti päivitetty ja varmistettu [8; 9.]

Tietokantaproxy (kuva 2) on asiakasohjelman ja tietokantaklusterin välissä toimiva välityspalvelin. Sen pääasiallinen tarkoitus on piilottaa useat replikaation alaisena olevat palvelimet yhden näkyvän verkko-osoitteen taakse. Tämä vähentää

asiakasohjelmien tiedon tarvetta tarjoamalla yhden verkko-osoitteen, jonne kaikki kyselyt voidaan tehdä. Tämä säilyttää replikaation skaalautuvuuden, virheidensietokyvyn ja dataturvallisuuden tarjoten kuitenkin yhden tietokantapalvelimen helppokäyttöisyyden ja yksinkertaisuuden. Tietokantaproxy voi tehdä kyselytaakan tasapainotusta jakamalla automaattisesti kyselyitä eri palvelimille.



Kuva 2. Klusteroidun tietokannan toiminta ilman tietokantavälityspalvelinta ja tietokantavälityspalvelimen kanssa

Työn käytännön osuuden aiheena oleva MaxScale-tietokantaproxy koostuu modulaarisista komponenteista, joiden välinen yhteys on toteutettu yleisen rajapinnan avulla. Komponentteja on erilaisia, ja niiden käyttö kohdistuu johonkin tiettyyn vaiheeseen kyselyiden välityksessä. Kyselyn reitityksestä vastaa reititysmoduuli, joka tekee lopullisen reitityspäätöksen ja välittää kyselyn oikealle tietokannalle. Ennen reititystä kysely välitetään filtterimoduulien läpi ja niillä on mahdollisuus muokata sen sisältöä tai tehdä muita toimia ennen varsinaista reitityspäätöstä. Yhteys asiakasohjelmaan on toteutettu omana moduulina, kuten on myös yhteys tietokantapalvelimiin. Tämä mahdollistaa tehokkaan ja räätälöidyn ratkaisun klusteroidun tietokannan hallinnointiin.

Asiakasohjelman tehdessä kyselyn MaxScale vastaanottaa sen, ohjaa sen filtterimoduulien käsiteltäväksi, jonka jälkeen reititysmoduuli tekee lopullisen reitityspäätöksen. Lähetettyään kyselyn eteenpäin tietokannalle reititysmoduuli jää odottamaan vastausta tietokannalta. Tämän vastauksen saatuaan reititysmoduuli palauttaa sen filtterimoduulien läpi asiakasohjelmalle [10.]

3 Klusteroidun SQL-tietokannan suorituskyky

Koska yleisin syy valita klusteroitu tietokanta on joko kyselyjen vasteajan parantaminen tai tietokannan saatavuuden ja kyselyjen suoritusvarmuuden parantaminen, on sen suorituskyky olennainen kysymys. Suorituskykyä ei kuitenkaan aina voi määritellä samalla tavalla, ja eri tilanteissa suorituskyky voi tarkoittaa monia eri asioita. Tämän työn näkökanta klusteroidun tietokannan suorituskykyyn keskittyy pääasiallisesti suuren kyselytaakan alla oleviin järjestelmiin. Esimerkkejä näistä ovat suurien verkkosivujen takana olevat tietokantapalvelinten klusterit, joihin voi kohdistua äärimmäisissä tapauksissa jopa 13 miljoonaa kyselyä sekunnissa [11.]

3.1 Suorituskyvyn määritelmä

Klusteroidun tietokannan suorituskyvyn voi määritellä usealla tavalla. Yleisin ja helposti mitattavin mittapuu suorituskyvylle on kyselyyn kuluva aika. Tämä on yksinkertaisin, helpoin ja yleisin tapa mitata jonkin sovelluksen tai palvelun toimintaa, ja parannukset tässä ovat huomattavissa välittömästi. Klusteroitujen tietokantojen tapauksessa tämä pitää myös paikkansa. Kyselyn nopea suoritus mahdollistaa suuremman kyselyjen määrän suorittamisen pienemmässä ajassa.

Kyselyn nopean suorittamisen lisäksi kyselyn varma suoritus on osa klusteroidun tietokannan suorituskykyä. Täten täydellinen suorituskyky vastaisi tilannetta, jolloin operaatiot tietokantaan eivät ikinä epäonnistuisi muista kuin asiakaspuolen käyttäjävirheistä. Käytännössä tämä ei kuitenkaan ole mahdollista, joten yksi ratkaisu tähän on klusteroida tietokanta. Master-Slave-replikaatiossa tämä kasvattaa lukukyselyjen suorituksen varmuutta kasvattamalla saatavilla olevien palvelimien määrää ja Multi-Master-replikaatiossa kaikkien kyselyjen suoritusvarmuus kasvaa. Tietokannan saatavuus on suurempi tekijä palveluissa, joissa korkea saatavuus on

tärkeää. Kyselyt voivat olla hitaita ja ne voivat epäonnistua, mutta palvelun tulisi silti olla saatavilla. Esimerkkejä tämänkaltaisista palveluista ovat pilvipalvelut, joiden varassa suuri osa moderneista tietotekniikan palveluista toimii.

Näistä kahdesta tekijästä voidaan muodostaa klusteroidun tietokannan suorituskyvylle määritelmä, jonka puitteissa suorituskykyyn vaikuttavia tekijöitä tässä työssä voidaan analysoida: klusteroidun tietokannan suorituskyky on kyselyn nopeuden ja tietokannan saatavuuden yhdistelmä.

3.2 Kyselyn nopeuteen vaikuttavat tekijät

Yksinkertaisimmat tekijät, jotka vaikuttavat kyselyn vasteaikaan, ovat tietokantapalvelimen laitteisto. Useimmat tietokannat pystyvät hyödyntämään moniajoa hyvinkin tehokkaasti, joten suoritinten ja suoritinytimien määrä on suoraan verrannollinen sen kyselyjen vasteaikaan. Toinen laitteistotason tekijä on muistin määrä. Levyjen luku- ja kirjoitusoperaatiot ovat hitaita operaatioita verrattuna muistissa suoritettuihin operaatioihin, joiden takia myös muistin määrä vaikuttaa yksittäisen palvelimen suorituskykyyn. Tämä johtuu suurimmalta osin siitä, että erinäiset taulut ja indeksit voidaan sijoittaa suoraan muistiin fyysisen median sijaan. Kuten Reinsel, et al. toteavat, nopeaan Flash-muistiin perustuvat SSD-kiintolevyt ovat yleistymässä palvelimissa. Nämä nopeuttavat lukuoperaatioita mahdollisesti jopa kymmenkertaisesti, vähentäen huomattavasti levyoperaatioon kuluvaa aikaa. Tästä seuraa suora riippuvuus suorituskyvyn ja SSD-kiintolevyjen välillä, varsinkin lukevien operaatioiden osalta [3].

Laitetasosta seuraava askel ylöspäin on verkkoyhteys. Sen nopeus vaikuttaa kyselyjen nopeuteen ja sen varmuus taas kyselyjen suorituksen varmuuteen. Vaikka tämä on tekijänä klusterin suorituskykyyn, ei sen vaikutusta yleensä lasketa kyselyn suoritusnopeudessa, koska se voidaan laskea osaksi infrastruktuuria. Siksi se on ympäristötekijä. Sijoittamalla palvelimet lähemmäksi asiakasohjelmaa vähennetään tarvittavien verkkohyppyjen määrää. Tämä on kenties yksi suurimmista tekijöistä kyselytaakan lisäksi, mikä vaikuttaa kyselyn nopeuteen, ja fyysisen etäisyyden minimoiminen lähimpään palvelimeen on hyvin tärkeää.

Varsinaisen klusterin tasolla suorituskykyyn vaikuttaa valittu replikaation tyyppi ja verkon topologia. Master-Slave-replikaation tapauksessa lukevat kyselyt ovat nopeita, koska yhteys Slave-palvelimeen on huomattavasti helpommin saatavilla kuin Master-palvelimeen olettaen, että Slave-palvelimia on ainakin yksi. Jos Master-palvelimia on vain yksi, kirjoittavien kyselyiden suorittaminen on huomattavasti lukevien kyselyiden suorittamista hitaampaa. Tämä johtuu pääosin siitä, että kyselyjen vaikutukset on kopioitava kaikkiin Slave-palvelimiin replikaation eheyden säilyttämiseksi. Master-Slave-replikaation yleisin käyttötarkoitus onkin suuren lukutaakan alla olevissa palveluissa, esimerkiksi tietovarastoissa. Näissä käytännön raja lukevien kyselyiden osalta on Slave-palvelimien määrä. Näiden palvelimien lisääminen ei kuitenkaan tuota lineaarista suorituskyvyn parannusta.

Multi-Master-replikaatiossa, jossa joko kaikki tai useampi kuin yksi tietokanta on Master-palvelin, kirjoittavat kyselyt ovat verrattain nopeita. Tämä yleensä johtuu pienemmästä taakasta yksittäiseen palvelimeen. Jos datan eheys ei ole välttämätöntä, voidaan käyttää asynkronista Multi-Master-replikaatiota. Tässä tapauksessa kaikkien solmujen ei tarvitse olla päivitettyjä, jotta niihin voidaan tehdä kelvollisia kyselyjä, joka johtaa nopeampaan, mutta vähemmän luotettavaan, toimintaan.

3.3 Keinoja parantaa suorituskykyä

Yksinkertaisin tapa parantaa tietokantaklusterin suorituskykyä on lisätä yksittäisen palvelimen suorituskykyä. Tämä parannus suorituskyvyssä on helposti huomattavissa, kun tietokanta-palvelinten määrä klusterissa on pieni ja yksittäisen palvelimen suorituskyky on merkittävä tekijä kyselyn nopeudessa. Fyysisen laitteiston päivityksellä on kuitenkin rajansa ja tietyssä pisteessä voi olla kannattavampaa hankkia lisää palvelimia paremman laitteiston sijaan.

Tietokantaklusterin tasolla suorituskykyä voidaan parantaa usealla eri tavalla. Yksi näistä on palvelinten määrän kasvattaminen. Lukukyselyiden osalta Slave-palvelinten määrä on käytännössä suoraan verrannollinen suorituskykyyn. Lisäämällä Slave-palvelinten määrää voidaan kasvattaa palveltavaa kyselytaakkaa ja täten parantaa tietokannan suorituskykyä. Tämä nähdään varsinkin, kun käytetään tietokantavälityspalvelinta, joka tekee kyselytaakan tasapainottamista Slave-palvelinten välillä. Tämä voisi olla suositeltava tapa parantaa klusterin suorituskykyä niin

sanotuissa tietovarastoissa, joiden sisältöä ei normaalisti päivitetä ja joihin kohdistuu monimutkaisia lukuoperaatioita. Tämä ei kuitenkaan onnistu tietokantaa muokkaavien kyselyiden osalta. Yleisin vastaus tähän ongelmaan yksittäisen tietokantapalvelimen tasolla on fyysisen laitteiston suorituskyvyn kasvatus. Tätä kutsutaan vertikaaliseksi skaalautuvuudeksi ja se saavuttaa nopeasti pisteen, jonka jälkeen saavutettavat hyödyt eivät ole oikeutettuja hintaansa nähden. Vastauksena tähän onkin horisontaalinen skaalaus. Esimerkiksi tilanne, jossa kaksi vähempitehoista palvelinta asetetaan kahden Master-palvelimen konfiguraatioon, voi olla suorituskyvyltään parempi kuin yhden palvelimen konfiguraatio, joka on laitteistoltaan kaksi kertaa tehokkaampi.

Replikaation ja sen toteutuksen valinta voi olla hyvinkin merkittävä tekijä suorituskyvyn kannalta, kun verkon koko, kyselyiden monimuotoisuus ja käyttäjämäärät kasvavat. PostgreSQL:n tarjoamat slony ja BDR replikaatiovaihtoehdot ovat hyvä esimerkki tästä.

Slony toimii kuten perinteinen Master-Slave-replikaatio ja onkin optimaalinen valinta, kun verkko ei ole monimutkainen. Tällöin kyselyt koostuvat pääosin lukevista operaatioista, ja käytössä on yksi Master-palvelin. Verkon koon kasvaessa kirjoituksien aiheuttaman verkkoliikenteen määrä kasvaa eksponentiaalisesti, joten hyvin suurten verkkojen kanssa, joissa kirjoittavat kyselyoperaatiot ovat suhteellisen suuri osa kyselytaakkaa, parempi vaihtoehto olisi BDR-replikaatio.

BDR, eli Bi-Directional Replication, on PostgreSQL:n tarjoama Multi-Master replikaatioteknologia, jossa jokainen tietokantasolmu on Master-tietokanta ja kaikki muutokset lähetetään muihin tietokantasolmuihin. Tässä replikaatiototeutuksessa jokainen paikallinen tietokantasolmu on sisäisesti eheä, ja klusteri kokonaisuutena on ”lopulta eheä”. Tällä ”lopullisella eheydellä” viitataan asynkroniseen tiedon päivitykseen käyttämällä solmukohtaista muutosten jonoa, ja joka saavuttaa klusterin laajuisen eheyden, kun kaikki muutokset on toteutettu.

Tämä teknologia mahdollistaa maantieteellisesti paikalliset tietokantasolmut ja kestävän tietokantaklusterin tarjoamalla laajan valikoiman varalla olevia tietokantapalvelimia. Maantieteellisesti lähellä olevat tietokantapalvelimet takaavat nopean kyselyn suorituksen, ja globaalisti tarjolla olevat varapalvelimet tarjoavat klusterille helpon keinon korkeaan saatavuuteen. Tämä on hyvin tehokas tapa toteuttaa klusteroitu korkean saatavuuden tietokanta, joka on kirjoittavien ja lukevien kyselyiden kannalta hyvin suorituskykyinen, jos tiedon eheydellä ja ajantasaisuudella ei ole kovin

suurta merkitystä. Esmerkkinä tämänkaltaisesta tilanteesta voisi olla hajautettu versionhallinta, jossa käyttäjät ottavat paikallisen kopion tiedosta, tekevät siihen muutoksia ja lopulta päivittävät versionhallinnassa olevan tiedon uuteen versioon. Tässä tiedon ristiriidat eivät ole ongelma, ja korkea saatavuus ja nopea kirjoituskyky ovat tärkeitä[13; 14.]

Partitiointi, eli tiedonosiointi, on keino parantaa tietokannan suorituskykyä jakamalla tietokannan sisältö useaan eri paikkaan. Partitioinnista on kahta eri tyyppiä, rivi- ja sarakekohtainen partitiointi. Rivikohtaisessa partitioinnissa tietokantataulun yksittäiset rivit ja sarakekohtaisessa partitioinnissa tietokantataulun sarakkeet sijaitsevat erillisillä osioilla. Rivikohtaisesta partitioinnista käytetään yleensä termiä sirpalointi.

”Sharding”, eli suomeksi sirpalointi, on tietokantaklustereiden kanssa käytettävä tekniikka, jossa tietokantojen sisältö jaetaan usealle eri fyysiselle palvelimelle. Näitä kutsutaan sirpaleiksi, joiden sisältöä ei löydy muista palvelimista. Esimerkiksi tietokantataulu, jossa on työntekijöiden tiedot aakkosjärjestyksessä, voi olla jaettu usean eri tietokantapalvelimen kesken siten, että työntekijät A:sta M:ään on sijoitettu yhdelle palvelimelle ja työntekijät M:stä Ö:hön on sijoitettu toiselle palvelimelle. Tämä parantaa kyselyyn kuluvaa aikaa varsinkin, jos tiedetään etukäteen, millaisia kyselyitä tähän tietokantaan tehdään ja mistä niitä tehdään. Ongelmia tämän tekniikan kanssa ilmenee, kun jokin tietokannoista ei olekaan saatavilla, kysely kohdistuu tietoon, joka sijaitsee kahdella sirpaleella, tietokannan skeemaa pitäisi päivittää tai transaktion sisällä olevat kyselyt kohdistvat useaan eri sirpaleeseen. Nämä ongelmat joudutaan yleensä ratkaisemaan loppukäyttäjän päässä ohjelmallisesti, koska sen tekeminen tietokannassa tai sirpalointia toteuttavassa välityspalvelimessa on monimutkainen toimenpide, joka on herkkä virheille. Sirpalointi on huonosti skaalautuva, koska sen toimivuus voi olla kiinni yhdestä tietokantapalvelimseta, joten sen käyttö suurissa verkoissa on todennäköisesti hyvin monimutkainen ja vaikeasti ylläpidettävä ratkaisu[5.]

Sarakekohtainen partitiointi mahdollistaa monimuotoisen datan tehokkaan käsittelyn. NoSQL-tyypiset ratkaisut käyttävät tämän tyyppistä partitiointia tehokkaasti hyväkseen, mutta tämä on myös mahdollista saavuttaa perinteisillä relaatiotietokannoilla. Jos tieto, jonka sarakkeissa olevan datan koko on suuri, jaetaan usean tietokantataulun kesken siten, että vähän käytetyt sarakkeet sijoitetaan yhteen tauluun ja usein käytetyt toiseen. Tällä mahdollistetaan usein käytetyn datan nopea saatavuus, mutta silti säilytetään

mahdollisuus harvemmin käytetyn datan helppoon saatavuuteen esimerkiksi näkymällä, joka yhdistää nämä kaksi taulua. Tämä partitiointityyppi kärsii kuitenkin samoista ongelmista kuin sirpalointi. Jos partitioitu tieto sijaitsee vain yhdellä palvelimella, ja tämä palvelin ei ole saatavilla, tietokannan sisältöä ei ole kokonaan saavutettavissa. Tämä ei välttämättä ole niin suuri ongelma kuin rivikohtaisen partitioinnin kanssa, koska jos tieto on jaettu sen käytön todennäköisyyden mukaan, voidaan parhaassa tapauksessa tietokantaa käyttää normaalisti, vaikka osa tiedosta ei ole saatavilla.

Yksi erilainen keino parantaa klusteroidun tietokannan suorituskykyä on vähentää verkkohyppyjen määrää kyselyitä tehdessä tai vähentämällä verkkoliikennettä kokonaisuudessaan, tehden osan tietokannasta tehtävissä operaatioissa jossain ohjelmistossa, joka on asiakkaan ja palvelinten välissä. Esimerkki tästä on MaxScale, joka hakee tietokannan käyttäjät etukäteen ja tekee osan käyttäjien autentikoinnista sisäisesti. Tällä vähennetään tarvittavaa verkkoliikennettä ja vähennetään täten kyselyyn kuluvaa aikaa.

Kaikillä näillä parannuskeinoilla on yhteistä se, että niistä mikään ei toimi kaikissa tilanteissa. Tämä on totta jo yhden tietokannan kanssa, mutta se korostuu varsinkin klusteroidussa tietokannassa, kun käyttäjien, palvelimien ja niistä aiheutuvien ongelmien määrä kasvaa. Joitain yleisiä suuntauksia suorituskyvyn suhteen voidaan tehdä, mutta niistä ei ole suurta käytännön hyötyä. Suurimmat parannukset suorituskyvyssä nähdään, kun suoritusympäristön tarpeisiin vastataan oikealla tavalla.

Jotta klusteroitu tietokanta toimisi mahdollisimman nopeasti ja luotettavasti, tulisi ensin tietää, kuka sitä tulee käyttämään ja miten, kuinka usein ja milloin sitä käytetään, minkälaisia odotuksia käyttäjillä on tiedon saatavuuden ja vasteajan suhteen ja kuinka paljon resursseja voidaan varata klusterin suorituskyvyn optimointiin. Vasta tämän jälkeen voidaan alkaa parantamaan tietokantaklusterin suorituskykyä korjaamalla huomatut ongelmakohdat. Käytännössä saatavilla olevat resurssit ovat yleisin syy rajoitteille tietokantaklusterin suorituskyvyssä, oli sitten kyseessä laitteistoon sijoitettavan rahan määrästä tai tietokannan ylläpitoon käytettävästä ajasta.

4 Palomuurisuodatin

Joissain tapauksissa yleisen tason suorituskyvyn parannukset eivät ratkaise käsillä olevaa ongelmaa, ja vaaditaan erikseen ongelmaa varten räätälöity ratkaisu. Esimerkkinä tällaista tapauksista ovat kolme erilaista tilannetta, joihin yksi mahdollinen ratkaisu on erillisen palomuurimoduulin kehitys.

Tapaus numero yksi on tietokanta, johon kohdistuu päivisin suuri kyselytaakka. Tähän tietokantaan kohdistuu myös kokoavia kyselyitä erinäisiä tilastoja ja raportteja varten. Nämä kokoavat kyselyt kestävät useita minuutteja ja hidastavat koko tietokannan muuta käyttöä huomattavasti. Tämä aiheuttaa tilanteita, jolloin päivisin tehtävät kokoavat kyselyt aiheuttavat huomattavia viiveitä tietokannan toiminnassa, ja suuri määrä kyselyitä odottaa suorittamista.

Tietokannan ylläpitäjien ei ole mahdollista varata yhtä tietokantapalvelinta pelkästään kokoavien kyselyiden käyttöön, joten haluttu toiminnallisuus on kyselyiden estäminen. Kokoavat kyselyt tietokantaan voitaisiin estää kokonaan, mutta tämän koetaan olevan liian aggressiivinen ratkaisu ongelmaan. Tämä myös poistaisi toiminnallisuutta kokoavia kyselyitä tekeviltä tahoilta, eikä tämä ole toivottua. Tietokannan haluttaisiin toimivan mahdollisimman nopeasti päiväsaikaan, mikä mahdollistaa kokoavien kyselyiden tekemisen muina aikoina.

Tapaus numero kaksi on esimerkki tilanteesta, jolloin tarvitaan tarkempaa kontrollia sallitusta kyselyiden määrästä. Tämän tapauksen ongelmana on tietokantaklusteri, johon kohdistuu äkillisiä, suuria kyselyitä, jotka hidastavat sen toimintaa. Kun suuri määrä kyselyjä tehdään hyvin pienellä aikavälillä, klusterin suorituskyky tippuu ja palvelun laatu tippuu alle halutun rajan. MySQL tarjoaa joitain keinoja rajoittaa sallittujen kyselyiden määrää per tunti, mutta sen tarjoama tarkkuus on liian pieni tähän tapaukseen. Jos sallittujen kyselyiden määrä on aina suhteellinen yhteen tuntiin, niin sen asettaminen alhaiseen arvoon voi estää kyselypiikkien muodostumisen, mutta se tuo mukanaan liikaa rajoitteita tietokannan normaaliin käyttöön. Tähän ongelmaan halutaan ratkaisu, joka mahdollistaa kyselyjen määrän rajoittamisen sekunnin tarkkuudella ja liian tiheiden kyselyiden estämisen joksikin tietyksi ajaksi.

Tapaus numero kolme on tilanne, jossa verkkosivuston käyttämään tietokantaan kohdistuu kyselyitä, jotka ovat automaattisesti generoituja jonkin ohjelmiston avulla.

Tietokantojen ylläpitäjä on huomannut tilanteita, jolloin päivitys- ja poistokyselyt joko tyhjentävät tai muokkaavat koko taulun sisältöä yhden rivin sijaan. Syynä tähän oli puute kyselyjä generoivassa ohjelmistossa, joka muodosti kyselyt ilman rajoittavaa

ehtoa. Esimerkiksi taulukon 1 viimeinen kysely oli muuttunut muotoon, joka johti kaikkien työntekijöiden lisäämisen ylläpitäjien tauluun. Tämä on huomattava turvallisuusriski, koska silloin kaikilla käyttäjillä olisi ylläpitäjien oikeude,t ja riski näiden oikeuksien väärinkäyttöön olisi hyvin suuri. Tämä on ongelma, joka vaatii ratkaisun, joka ei johda ohjelmistopäivityksiin asiakkaan päässä, mutta kuitenkin ratkaisee käsillä olevan ongelman.

Eheä kysely	Rikkoutunut kysely
UPDATE employees SET salary=5000 WHERE id=1	UPDATE employees SET salary=5000
DELETE FROM employees WHERE id=1	DELETE FROM employees
INSERT INTO administrators SELECT * FROM employees WHERE id=1	INSERT INTO administrators SELECT * FROM employees

Taulukko 1. Ohjelmiston muodostamia rikkoutuneita kyselyitä

Rikkoutuneiden kyselyiden estämiseksi halutaan asettaa kyselyille syntaksisia rajoitteita, joita haluttujen kyselyiden tulee noudattaa. Esimerkkinä kaikkien poisto- ja päivityskyselyissä tulee olla SQL-kielen WHERE-lauseke. Jos kyselyssä ei tätä ole, se tulisi estää.

Näiden kolmen esimerkkitapausten avulla kuvataan MaxScale-ohjelmistoa varten kehitettävän, palomuurina toimivan suodatinmoduulin suunnittelua, toteutusta ja toiminnan testausta. Suunniteltu toiminnallisuus kohdistetaan suurimmalta osin esimerkkitapausten ratkaisuun ja moduulin testauksessa käytettävä materiaali tulee ensin esittelemään esimerkkitapausten ongelmat käytännön tasolla jonka jälkeen

materiaalia käytetään analysoimaan palomuurisuodattimen kelpoisuutta kunkin ongelman ratkaisuun.

Moduulin ohjelmointikielenä käytetään C-kieltä. Suurimmat syyt ohjelmointikielen valinnalle on UNIX-pohjainen suoritusympäristö, ja koska MaxScale on kirjoitettu suurimmalta osin C-kielellä. Erillisiä ulkoisia kirjastoja ei tulla käyttämään, koska riippumattomuus suoritusympäristöstä on moduulille suotuisa ominaisuus ja koska palomuurisuodattimen tarkoitus on olla yksinkertainen ratkaisu tietynkaltaisiin ongelmiin.

4.1 Moduulin tavoitteet ja vaatimukset

MaxScale-moduulina toteutetun palomuurisuodattimen tavoitteena on tarjota lisäkerros toiminnallisuutta jo olemassa olevan käyttäjähallinnan päälle. Suodattimen ideana oli rikastaa SQL:n tarjoamaa GRANT-pohjaista käyttäjänhallintaa antamalla keinoja rajoittaa oikeuksia esimerkiksi yksittäisiin sarakkeisiin, tiettyä säännöllistä lauseketta vastaaviin kyselyihin tai estää kyselyt kokonaan tiettyinä kellonaikoina. Näillä pyritään mahdollistamaan uusia tapoja hallinnoida tietokantoja ja parantamaan suorituskykyä vähentämällä tarvittavien verkkohyppyjen määrää tai vähentämällä kyselytaakkaa estämällä tietynlaiset kyselyt.

SQL:n GRANT-pohjainen oikeuksien hallinta käyttää käyttäjäkohtaisia oikeuksia tietokantaobjekteihin ja tietokannan resursseihin. Sen avulla voidaan rajoittaa käyttäjien oikeuksia tietokantoihin, niiden tauluihin tai sarakkeisiin. Myös sallittuja operaatiotyyppejä voi hallita ja esimerkiksi pelkän lukuoikeuden antaminen johonkin tauluun tai tietokantaan on yleinen tapa estää virheellinen tietokannan käyttö. Muita hallittavia asioita on esimerkiksi salatun käyttäjäyhteyden vaatiminen. GRANT-pohjainen oikeuksien hallinta ei ole dynaamista, eikä se huomioi suoritusympäristöä tai ulkoista maailmaa millään tavalla. Tämän takia palomuurisuodattimen kaltaisia ratkaisuja on käytettävä, jotta tarvittava tietokannan oikeuksien dynaamisuus saavutetaan.

Käyttäjien kyselyiden rajoittaminen tiettyinä aikoina on esimerkki tilanteesta, jolloin SQL:n tarjoama GRANT-pohjainen oikeuksien hallinta on puutteellinen. Tällä hetkellä SQL-kieli ei tarjoa keinoja rajoittaa kyselyiden tekemistä tiettyinä aikoina, vaan tämä

joudutaan tekemään jonkun muun mekanismin kautta. Tämän opinnäytetyön tapauksessa tuo mekanismi oli opinnäytetyötä varten kehitetty palomuurisuodatin.

Palomuurisuodattimen pitää noudattaa tiettyjä vaatimuksia, jotka tulevat MaxScale-ohjelmiston puolelta. Moduulin tulee noudattaa MaxScalen tarjoamaa suodatin-rajapintaa, koska sen ei oleteta olevan pysyvä osa ohjelmistoa vaan lisäosan kaltainen moduuli. Näillä moduuleilla saavutetaan kunkin tapauksen tarpeiden mukainen räätälöity toteutus. Toinen vaatimus on sen toteutus C- tai C++-kielellä, koska tällä hetkellä MaxScale ei tarjoa keinoja käyttää muita kieliä. Muita tavoitteita moduulin toiminnalle on mahdollisimman asynkroninen suoritus ja pienin mahdollinen vaikutus muiden moduulien toimintaan ja MaxScalen suorituskykyyn. Näitä voi edistää optimoimalla tietorakenteita ja välttämällä raskaita operaatioita, kuten levyltä lukua.

Jotta palomuurisuodatin olisi toimiva ratkaisu ensimmäiseen esimerkkitapaukseen, pitää sen pystyä tunnistamaan joko tietyt kyselyn piirteet, tietokannan sarakkeet tai taulut, jotka johtavat tietokannan hidastumiseen. Kun palomuurifiltteri tunnistaa nämä kyselyt, tulee sen estää ne.

Jos kyselytaakan sisältö tiedetään tarpeeksi tarkasti, on säännöllinen lauseke yksinkertaisin tapa estää nämä kyselyt. Jos kyselyiden sisältöä ei kuitenkaan voida määritellä tarpeeksi tarkasti, mutta tiettyjä sen ominaisuuksia tiedetään, voidaan nämä tietyt ominaisuudet estää. Esimerkkejä näistä ominaisuuksista voi olla UNION-komennon käyttö tai kaikkiin tai johonkin tiettyyn sarakkeeseen kohdistuva kysely.

Esimerkkitapauksen numero kaksi vaatimus on kyky estää liian usein tapahtuvat kyselyt. Tämä on helpoin saavuttaa laskemalla jollain tietyllä aikajaksolla tapahtuvien kyselyiden määrä, ja jos se ylittyy, tulevat kyselyt kyseiseltä käyttäjältä estetään.

Kolmannen esimerkkitapauksen vaatimuksen täyttäminen vaatii palomuurisuodattimelta kyvyn havaita puutteita kyselyssä. Tämä on muista poikkeava vaatimus siinä mielessä, että se ei estä kyselyä, kun jokin tekijä on kyselyssä, vaan silloin kun siitä puuttuu jokin tietty tekijä. Koska MaxScale käyttää sisäisissä reitityslogiikassaan sisäänrakennettua MySQL-tietokantaa, voidaan sen tarjoamaa leksikografista rakennetta käyttää hyväksi kyselyn muodollista sisältöä tarkasteltaessa. Jos tätä vaihtoehtoa ei olisi käytettävissä, toinen mahdollisuus olisi käyttää säännöllistä lauseketta, mutta sen suorituskyky olisi heikompi, koska pitkien kyselyiden

läpikäyminen säännöllisen lausekkeen avulla on hidasta, ja toimivuus ei olisi välttämättä niin varmaa.

4.2 Palomuurisuodattimen toteutus

Kehitys aloitetaan toteuttamalla suodatin-rajapinnan vaatimat funktiot ja toiminnallisuus. Tämän jälkeen toteutetaan kyselyn estäminen ja virhepakettien lähettäminen asiakasohjelmalle, jonka jälkeen varsinaiset säännöt toteutetaan.

Suodatin valitsee tarkistettavat säännöt käyttäjänimen ja verkko-osoitteen perusteella. Tämä tehdään samalla periaatteella kuin käyttäjäoikeuksien tarkistaminen SQL-tietokannoissa. Kun tarkin verkko-osoitteen ja käyttäjänimen määritelmä löydetään, käytetään sitä vastaavia sääntöjä ja tarkistetaan, onko kysely joko kaikkien sääntöjen tai vain osan vastainen. Kyselyn ollessa sääntöjen vastainen, sen reititys keskeytetään ja asiakasohjelmalle lähetetään virhepaketti, joka kertoo, minkä säännön vastainen kysely oli.

Sääntöjen syntaksi suunnitellaan yksinkertaiseksi ja helposti laajennettavaksi. Tämä tehdään, jotta suodatinta voidaan tarvittaessa muokata jotain tiettyä ongelmaa varten. Säännöt noudattavat yksinkertaista rakennetta, jossa säännölle annetaan nimi, tarkistettavat ehdot ja tehtävä operaatio, jos ehdot on saavutettu. Palomuurin tarkoituksen ollessa liikenteen estäminen, on tehtäviä operaatioita vain kaksi: kyselyn estäminen tai hyväksyminen.

Ensimmäisen esimerkkitapauksen vaatimukset päätetään täyttää mahdollistamalla säännöllisten lausekkeiden käyttäminen. Nykyään C++-kieli tarjoaa standardoidun keinon tämän toteutukseen, mutta MaxScalen ollessa UNIX-ympäristöihin suunniteltu sovellus, päädyttiin kuitenkin GNU C-kirjaston tarjoamaan regex-toteutukseen. Tämän uskotaan olevan yksinkertaisin keino, jolla mahdollistetaan mahdollisimman laaja sääntöjen ilmaisu. Toinen vaatimus tähän esimerkkitapaukseen oli sääntöjen voimassaolo vain tiettyinä aikoina. Koska tämä on hyvin yleinen ja laajalti kysytty vaatimus, toteutetaan se filtterin sisäisenä rakenteena, joka määrittää linkitetyn listan aikajaksoja. Tämän rakenteen avulla määritellään aikajaksot, jolloin sääntö on aktiivisena ja mahdollistetaan joustavan sääntöympäristön muodostaminen[15.]

Muina mahdollisina keinoina, jolla kyselyitä voitaisi rajata, mainitaan tiettyyn sarakkeeseen kohdistuvat kyselyt ja jokerimerkin käyttö. Vaikka tämän esimerkkitapauksen ongelmanratkaisun kanssa käytetään säännöllistä lausetta, toteutetaan tiettyyn sarakkeisiin kohdistuvan kyselyn rajaus ja jokerimerkin tunnistus niiden yleisen hyödyllisyyden vuoksi. Ne myös mahdollistavat vaihtoehtoisia tapoja tunnistaa raskaat kyselyt ja tuovat mahdollisia helppoja ratkaisuja tuleviin samankaltaisiin ongelmiin.

Kyselyn kohteena olevat sarakkeet saadaan selville leksikografisen rakenteen avulla. Vertaamalla näitä arvoja sääntöön tallennettuihin arvoihin saavutetaan sarakekohtainen oikeuksien hallinta. Tämä on kuitenkin liian raskas toteutus suurien kyselyiden kanssa. Jos kysely kohdistuu kymmeniin tauluihin ja vielä suurempaan määrään sarakkeita, voi kyselyn laskennallinen vaativuus kasvaa eksponentiaalisesti. Ongelmaksi tämä muodostuu jo pienissä tietokannoissa, joten sen käyttö tuotantotason ohjelmistoissa on epätodennäköistä.

Verrattuna edelliseen vaihtoehtoon Jokerimerkin toteutus on hyvin kevyt. Koska sisäisen tietokannan leksikografinen rakenne pitää sisällään kaikki sarakkeet, yksinkertaisin toteutus on vain käydä ne läpi ja tarkistaa, jos joku niistä vastaa jokerimerkkiä. Laskennalliselta teholtaan tämä johtaa lineaariseen läpikäyntiaikaan.

Toisen esimerkkitapauksen ongelma ratkaistaan kehittämällä rakenne, joka pitää sisällään mitattavan aikavälin, sallittujen kyselyiden määrän, ensimmäisen mitattavan kyselyn ajankohdan, tapahtuneiden kyselyiden määrän, ajan, jolloin sallittujen kyselyjen määrä ylitettiin, ja ajan pituuden, jona tulevat kyselyt estetään, jos sallittujen kyselyiden raja ylitetään. Kun kysely saapuu palomuurisuodattimeen, tarkistetaan, onko viime kyselystä kulunut enemmän aikaa kuin mitattava aikaväli on. Jos kysely tapahtuu mitattavan aikavälin sisällä, kasvatetaan tapahtuneiden kyselyiden määrää yhdellä. Jos tämä arvo ylittää asetetut rajat, estetään kysely ja kaikki käyttäjältä tulevat kyselyt estoajan ajaksi.

Kuvassa 3 näkyvä struct-rakenne pitää sisällään kyselyn nopeutta kuvaavat tiedot. Tässä QUERYSPEED-rakenteessa first_query-kentässä pidetään aika, jolloin ensimmäinen kysely tehtiin. Tällä määritellään mitattavan aikavälin alku. Mitattavan aikavälin pituus saadaan vähentämällä kyselyhetkellä mitatusta ajasta tämä ensimmäisen kyselyn aika. Jos tämä aikaväli on pienempi kuin perion-kentässä oleva

arvo, kasvatetaan count-kentän arvoa yhdellä. Count-kenttä pitää sisällään mittausaikavälin sisällä tehtyjen kyselyiden määrän Period-kenttä pitää sisällään mittausaikavälin pituuden ja limit-kenttä pitää sisällään kyselyjen lukumäärän ylärajan. Jos count-kentän arvo ylittää limit-kentän arvon ja mitattava aikaväli on pienempi kuin period-kentän arvo, tulkitaan kyselynopeus liian nopeaksi. Tämä rakenne on suorituskyvyltään tehokas ja toteutukseltaan yksinkertainen. Koska se käyttää pieniä perustyypin kenttiä, joita on nopea verrata ja kasvattaa ja jotka vievät vähän muistia, taataan pieni muistijalanjälki ja rakenteen nopea käsittely.

```
typedef struct queryspeed_t{
    time_t first_query;
    time_t triggered;
    double period;
    double cooldown;
    int count;
    int limit;
}QUERYSPEED;
```

Kuva 3. Kyselynopeutta kuvaava rakenne

Tämän rakenteen suhteellisen yksinkertainen toteutus mahdollistaa hyvin tarkasti hallittavissa olevan kyselynopeuden rajoittamisen hyvin tehokkaasti. Tämä rakenne ja sen toiminta ovat käytännössä vain tarkempi vastine MySQL-tietokannassa olevan kyselyiden nopeuden rajoittamiselle. Koska käytössä on sekunnin tarkkuudella mitattu kyselynopeus MySQL:n tunnin sijaan, voidaan kyselynopeutta rajoittaa hyvin erilaisissa ympäristöissä. Tällä mahdollistetaan esimerkiksi DoS-hyökkäyksen esto. DoS-hyökkäykset, eli palvelunesto-hyökkäykset, ovat verkon kautta tehtäviä hyökkäyksiä, jossa jokin palvelu on jättimäisen kyselytulvan alla ja tämä palvelu ei pysty käsittelemään kasvanutta kyselytaakkaa tarvittavan hyvin.

Kyselyiden estämisen sijaan voitaisiin kyselyt asettaa jonkinmuotoiseen jonoon. Tämä ei kuitenkaan ole palomuurin tarkoitus eikä täten myöskään tämän moduulin tarkoitus, joten tätä toiminnallisuutta ei toteuteta. Tämä estää myös mahdolliset tilanteet, joissa suuren kyselytulvan alla olevan MaxScale-ohjelmisto pitäisi sisällään suuria määriä odottavia kyselyitä. Tämä toiminnallisuus on myös jossain määrin välityspalvelimen alkuperäisen tarkoituksen vastainen, joka on tiedon nopea ja läpinäkyvä välitys.

Kyselytyyppien mukaan tehtävä kyselyn tarkistaminen on toinen hyvin yleinen tarve säännöille. Tämä toiminnallisuus saadaan aikaan, kun sisäisen tietokannan

leksikografisen rakenteen kautta saadaan selville, onko kyseessä päivitys-, poisto-, luonti- vai valintakysely. Asettamalla sääntöön tämän rajoitteen voidaan luoda tarkkoja rajoja sääntöjen avulla. Kyselyn saapuessa kyselyn tyyppiä verrataan säännössä olevaan, ja jos se vastaa säännössä olevaa tyyppiä, kysely tarkistetaan.

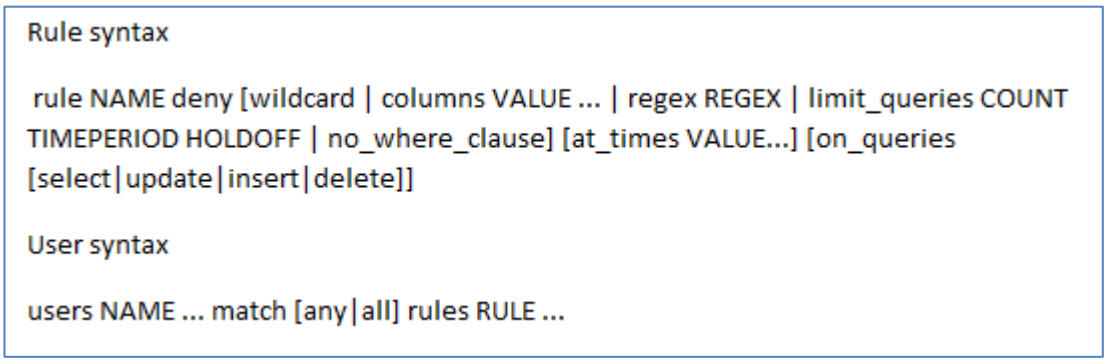
Kolmannen esimerkkitapauksen vaatimukset täytetään käyttämällä edellä mainittua kyselytyyppirakennetta ja sisäisen tietokannan tarjoamaa kykyä tarkistaa, onko kyselyssä jokin rajaehto. Jos rajaehtoa ei ole ja kyselyn tyyppi on sama kuin säännön kyselytyyppi, kysely estetään. Tällä saadaan aikaan lisäturvallisuutta tarjoava kerros, mikä estää mahdolliset inhimilliset, kuten myös ohjelmalliset, virheet.

4.3 Säännöt

Palomuurimoduulin käyttämät säännöt määritellään käyttämällä yksinkertaista syntaksia. Syntaksin avulla määritellään säännöt ja listat käyttäjistä, joihin näitä sääntöjä kohdistetaan. Yksittäinen sääntö määritellään käyttämällä avainsanaa “rule”, jonka jälkeen tulee säännön nimi. Tämän jälkeen määritellään tehtävä operaatio, joka on palomuurin tapauksessa kyselyn estäminen. Tämä tunnistetaan avainsanasta “deny”, jonka jälkeen määritellään, minkä muotoinen ehto sille asetetaan. Ehtojen, ja niiden määritelmien jälkeen, tulee kaikille säännöille yhteiset vaihtoehtoiset rajoitteet. Nämä rajoitteet voivat olla joko jokin tietty aika, jolloin sääntö on aktiivisena, tai lista SQL komennoista, joiden kohdalla sääntö on aktiivisena. Mahdollisia ehtoja ovat säännöllinen lauseke, kyselyn nopeuden rajoitus, WHERE-komennon olemassaolo, sarakkeen nimi ja jokerimerkin käyttö.

Käyttäjät määritellään avainsanalla “users”. Tämän jälkeen tulee lista käyttäjän ja verkko-osoitteen muodostamista pareista, joiden kanssa voidaan käyttää jokerimerkkinä prosenttimerkkiä, samoin kuten SQL:n kanssa. Verkko-osoitteiden jälkeen tulee avainsana “match”, jonka jälkeen tulee joko “any” tai “all”. Jos käytetään ensimmäistä vaihtoehtoa, kysely estetään, jos mikä tahansa säännöistä vastaa kyselyä. Jälkimmäisen kanssa kaikkien sääntöjen pitää vastata kyselyä. Tämän jälkeen avainsanalla “rules” määritellään lista sääntöjen nimiä, jotka tarkistetaan käyttäjän tehdessä kyselyn.

Esimerkiksi käyttäjälle “John”, joka ottaa yhteyttä mistä tahansa verkko-osoitteesta, asetetaan säännöt “regex_rule” ja “wildcard_rule” määrittelemällä lauseke “users John@% match any rules regex_rule wildcard_rule ”.



Kuva 4. Sääntöjen ja käyttäjien määrittämiseen käytetty syntaksi

Näistä säännöistä on lisää käytännön esimerkkejä seuraavassa kappaleessa, joka käsittelee moduulin konfiguraatiota.

4.4 Moduulin konfigurointi ja ylläpito

Moduulin säännöt sijaitsevat erillisessä tekstitiedostossa, jonka sijainti syötetään MaxScalen omaan konfiguraatiotiedostoon. Tämä tiedosto pitää sisällään kaikki säännöt ja käyttäjät, joihin niitä käytetään.

```
[hint]
type=filter
module=fwfilter
rules=/home/admin/rules.txt
```

Kuva 5. Palomuurimoduulin konfiguraatio MaxScalen konfiguraatiotiedostossa

Palomuurimoduulin säännöt sijaitsevat erillisessä tiedostossa, koska tämä vähentää MaxScalen ja palomuurimoduulin välistä riippuvuutta samalla mahdollistaen yhtenäisen säännöstön usean eri MaxScalen välillä. Esimerkiksi jokin yritys voisi määritellä joitain globaaleja sääntöjä, jota kaikkien MaxScalen instanssien pitää noudattaa. Sijoittamalla tämän yhteen paikkaan, josta kaikki MaxScalet pääsevät siihen käsiksi, saadaan aikaan yhdenmukainen ja helposti muokattava säännöstö.

Kuvan 6 sääntötiedostossa on määritelty viisi sääntöä. Näistä ensimmäinen määrittelee säännön ”peak_hour_rule1”, joka kieltää jokerimerkin käytön, ja toinen määrittelee säännön ”peak_hour_rule2”, joka puolestaan asettaa rajoituksen kyselynopeudelle, 500 kyselyä kahden sekunnin sisällä. Ensimmäinen sääntö estää vain yksittäisiä kyselyitä, kun taas toinen sääntö estää kaikki kyselyt seuraavan neljän sekunnin ajan, jos kysely vastaa sääntöä. Nämä molemmat säännöt ovat aktiivisia alkaen kello 17:00:00 ja loppuen kello 17:30:00. Loput säännöistä asettavat rajoitteita kyselynopeudelle, joissa estoaika kasvaa kyselynopeuden kasvaessa. Näistä ensimmäinen, ”limit1”, estää kyselyt seuraavan sekunnin ajaksi, jos kyselynopeus ylittää kymmenen kyselyä sekunnissa, toinen sääntö, ”limit2”, estää kyselyt kahden sekunnin ajaksi, jos kyselynopeus on suurempi kuin 150 kyselyä sekunnissa ja kolmas, ”limit3”, estää kyselyt neljän sekunnin ajaksi, jos kyselynopeus ylittää 250 kyselyä sekunnissa. Nämä kolme sääntöä ovat koko ajan aktiivisia. Näitä sääntöjä kohdistetaan kahteen eri käyttäjään. Näistä ensimmäinen, ”maxtest@%”, tarkoittaa käyttäjää maxtest, joka ottaa yhteyden mistä tahansa verkko-osoitteesta. Tähän käyttäjään kohdistetaan kolme sääntöä, ”limit1”, ”limit2” ja ”limit3”. Jos mikä tahansa näistä säännöistä on käyttäjän tekemää kyselyä vastaan, estetään käyttäjän kyselyt säännön perusteella. Tässä tapauksessa kyseessä voi olla joko yhden, kahden tai neljän sekunnin esto. Toinen käyttäjä, johon sääntöjä kohdistetaan, on ”%@%”. Koska prosenttimerkkiä käytetään jokerimerkkinä käyttäjänimiä ja verkko-osoitteita jäsennettäessä, tämä käyttäjä vastaa kaikkia käyttäjiä kaikista verkoista. Tämä on keino asettaa globaaleja sääntöjä ja tässä tapauksessa sitä käytetään kahden ensimmäisen säännön kohdistamiseen kaikkiin käyttäjiin. Näin saadaan aikaan vain ruuhkatuntien aikana aktiivisena oleva säännöstö kaikille käyttäjille, joka parantaa tietokannan suorituskykyä ja toimivuutta.

```
rule peak_hour_rule1 deny wildcard at_times 17:00:00-17:30:00
rule peak_hour_rule2 deny limit_queries 500 2.0 4 at_times 17:00:00-17:30:00
rule limit1 deny limit_queries 10 1.0 1
rule limit2 deny limit_queries 150 1.0 2
rule limit3 deny limit_queries 500 2.0 4
users maxtest@% match any rules limit1 limit2 limit3
users %@% match any rules peak_hour_rule1 peak_hour_rule2
```

Kuva 6. Palomuurimoduulin säännöt määrittelevä tiedosto

4.5 Palomuurimoduulin testaus ja soveltuvuuden arviointi

Palomuurisuodattimen toimivuuden testausta varten kehitetään erilliset testit jokaista testitapausta kohden. Testien kyselyt on muodostettu pyrkien emuloimaan käytännön toimintaympäristöä mahdollisimman realistisesti. Näitä kyselyitä käytetään toimivuuden testaukseen, kuten myös varsinaisen suorituskyvyn testaukseen. Toimivuutta testaavat testit käyttävät MaxScalen mukana tulevaa suodatinmoduulien testausohjelmaa, joka on tarkoitettu yksinkertaiseen toimivuuden testaamiseen. Tälle ohjelmalle syötetään SQL-kyselyitä, jotka se syöttää suodatinmoduuleille. Lopulta ohjelma tulostaa lopullisen kyselyn sisällön erilliseen. Tämän avulla moduulin toimivuus voidaan testata ilman ulkoisia tekijöitä, jotka voisivat vaikuttaa testauksen lopputulokseen. Tämän avulla voidaan myös testata pelkän moduulin vaikutus ilman kyselyn vaikutusta suorituskykyyn.

Suorituskyvyn testaukseen käytetään neljää samanaikaista MariaDB SQL-palvelinohjelman instanssia, jotka on konfiguroitu Master-Slave replikaatioon. Näissä on tietokanta ”test”, jossa on taulu t1. Taulu pitää sisällään kaksi kenttää: kentän ”id”, joka on automaattisesti kasvava avainarvo, ja kentän ”value”, joka on kokonaisluku. Tämä taulu on täytetty testausta varten noin miljoonalla rivillä dataa. Tällä saadaan luotua sopiva toimintaympäristö, jossa testit voidaan suorittaa. Tätä tietokantaa ja MaxScale-ohjelmistoa ajetaan samalla tietokoneella verkkoliikenteen minimoimisen takia. Tällä saavutetaan helposti muokattava testausympäristö ja helppo testien suorittaminen. Haittapuolina tällä testausympäristöllä on oikean verkkoliikenteen ja varsinaisten palvelinten puute, joka voi luoda vääristyneitä testituloksia verrattuna varsinaiseen suoritusympäristöön.

Työkaluina toiminnallisen testauksen tulosten ajonaikaiseen varmistukseen käytetään GDB-virheenjäljitintä. Tällä voidaan tulkita testimateriaalin vaikutus, tarkistaa looginen toimivuus ja huomata mahdolliset sivuvaikutukset, joita ei muuten näe helposti [4].

Ensimmäisen esimerkkitapauksen testikyselyt (kuva 6) on muodostettu kuvaamaan suurta määrää pieniä ja nopeita kyselyitä, joiden välissä tapahtuvat suuret kyselyt. Pienten kyselyjen emulointi tehdään valitsemalla yksi rivi testitaulusta ja suurten kyselyiden valitsemalla koko taulun sisältö kaksi kertaa SQL-kielen UNION-komennon

avulla. Tällä saadaan aikaan hyvin suuria viiveitä, jos taulun koko on myös hyvin suuri. Toimivuuden kannalta näillä ei ole kuin pinnallinen vaikutus, mutta varsinaisissa suorituskyvyn mittauksissa tämä nähdään huomattavina viiveinä kyselyn suorituksessa jo miljoonan rivin kokoisissa tauluissa.

```
select id from t1 limit 1;
select id from t1 union select id from t1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 union select id from t1;
select id from t1 limit 1;
select id from t1 limit 1;
```

Kuva 7. Ensimmäisen esimerkkitapauksen testausta varten muodostetut kyselyt

Odotettu tulos saavutettiin ja pelkästään kyselyt ilman UNION-komentoa pääsivät läpi. Toimivuus varmistettiin myös virheenjäljittimen avulla tutkimalla palautettuja arvoja. Kaikki kyselyt, jotka sisälsivät UNION-komennon, tunnistettiin säännöllisen lausekkeen avulla, jona jälkeen ne estettiin.

```
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
```

Kuva 8. Ensimmäisen testitapauksen lopputulos

Toisen esimerkkitapauksen kyselyt (kuva 8) valittiin kyselyiksi, jotka ovat nopeita suorittaa. Kun valitaan vain yksi rivi ilman erityisiä rajoitteita, saadaan aikaan nopea kysely. Tällä voidaan helposti säädellä tehtyjen kyselyiden määrää, kun kyselyt suoritetaan tietyin aikavälein. Näin voidaan testata toisen esimerkkitapauksen toimivuus ja suorituskyky. Toimivuuden testausta varten sallittu kyselyiden määrä on viisi kyselyä sadan sekunnin sisällä. Jos tämä raja ylitetään, estetään kyselyt seuraavan kahden sadan sekunnin ajaksi. Tällä voidaan varmistaa moduulin toimivuus laskemalla läpi päässeiden kyselyiden määrä. Jos läpi päässeiden kyselyiden määrä on asetettujen rajojen sisällä, toimivuus voidaan todeta riittäväksi.

```
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
```

Kuva 9. Toisen esimerkkitapuksen testikyselyt

Toisen esimerkkitapauksen tulokset vastasivat odotettuja arvoja ja vain viisi ensimmäistä kyselyä pääsi läpi.

```
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
select id from t1 limit 1;
```

Kuva 10. Toisen testitapauksen lopputulos

Kolmatta esimerkkitapausta varten käytetään kyselyitä, jotka päivittävät yhtä saraketta taulussa (kuva 10). Puolet kyselyistä kohdistuvat riviin, jonka sarakkeen ”id” arvo on yksi, ja puolet kohdistuvat kaikkiin taulussa oleviin riviin. Tiedon päivittäminen poistamisen sijaan tehdään, jotta suorituskyvyn mittaaminen onnistuu testaustietokannassa ilman, että testaustietokannan sisältö jouduttaisiin luomaan joka kyselysarjan jälkeen uudestaan. Tämä valinta tehtiin myös sen takia, että virhetilanteesta johtuvat seuraukset nähdään helposti ja suorituskyvyn mittaaminen, ja sen parannukset, voidaan tehdä mittaamalla kyselyyn kulunut aika. Toimivuuden kannalta hyväksytty tulos on saavutettu, jos vain puolet kyselyistä pääsee läpi. Suorituskyvyn kannalta tämän kyselysarjan tulisi saavuttaa huomattavasti parempi suoritusaika palomuurisuodattimen kanssa verrattuna kyselysarjan suoritukseen ilman sitä. Koska testitietokannassa oleva taulu pitää sisällään noin miljoona riviä, vie kaikkien sen rivien päivittäminen huomattavasti enemmän aikaa kuin vain yhden rivin päivittäminen.

```
update t1 set value=1 where id=1;
update t1 set value=1;
update t1 set value=2 where id=1;
update t1 set value=2;
update t1 set value=3 where id=1;
update t1 set value=3;
update t1 set value=4 where id=1;
update t1 set value=4;
```

Kuva 11. Kolmannen esimerkkitapauksen testikyselyt

Tämänkin esimerkkitapauksen tulokset vastasivat odotettuja arvoja ja vain kyselyt, joissa id-kentän arvo oli yksi pääsivät läpi. Tämä varmistettiin tutkimalla tietokannan taulun sisältöä suoralla yhteydellä tietokantaan. Vain kentät, joiden id-kentän arvo oli ollut yksi, muuttivat arvoaan.

```
update t1 set value=1 where id=1;
update t1 set value=2 where id=1;
update t1 set value=3 where id=1;
update t1 set value=4 where id=1;
```

Kuva 12. Kolmannen esimerkkitapauksen läpipäässeet kyselyt

Näiden testitulosten avulla moduuli todettiin toimivaksi näiden esimerkkitapauksien kohdalla ja myös yleisellä tasolla.

4.6 Ulospäin näkyvä käyttäytyminen

MaxScale ei näy loppukäyttäjälle kovinkaan helposti. Joitain poikkeuksia tähän on, kuten sessiomuuttujien käyttämisen osittaiset rajoitteet joissain kyselyissä. MaxScalen tapauksessa loppukäyttäjänä voi olla asiakas, joka ottaa yhteyden suoraan MaxScaleen, tai esimerkiksi jokin muu ohjelma, joka käyttää SQL-kyselyitä.

Palomuurimoduuli näkyy loppukäyttäjälle vain, jos jotain asetettua sääntöä rikotaan. Tässä tapauksessa loppukäyttäjä saa virheestä kertovan SQL-paketin, jonka viestinä on syy kyselyn estämiselle. Tämä toteutettiin, jotta loppukäyttäjälle tarjottaisiin mahdollisimman tarkkaa tietoa kyselyn estämisestä. Jos räätälöityä virhepakettia ei

käytettäisi, jouduttaisiin käyttämään esimerkiksi MySQL:n tarjoamia virhepaketteja, joka voisi johtaa väärinkäsityksiin virhetilanteissa.

MaxScalen ideologiana jossain määrin on tarjota läpinäkyvä toteutus tietokantavälityspalvelimesta, jonka voi ottaa käyttöön jo olemassa olevaan tietokantaklusteriin. Nykyisen kehityksen suunta on menossa tähän suuntaan ja tästä syystä palomuurimoduulin, ja MaxScalen, loppukäyttäjälle näkyvä käyttäytyminen on parhaassa tapauksessa olematon.

Ylläpitäjälle palomuurifiltterin käytös näkyy MaxScalen lokitiedostoista. Viestit sisältävät tiedon siitä kuka teki kyselyn, koska kysely tehtiin ja minkä takia kysely estettiin. Näin saadaan suhteellisen helposti selville tapaukset, joissa on tehty kyselyitä, jotka ovat sääntöjen vastaisia.

Palomuurimoduulin läpi kulkee paljon tietoa, jota analysoidaan usealta eri kannalta, ja tämä tieto on mahdollisesti tilastollisessa mielessä arvokasta. Palomuurimoduuli ei kuitenkaan suorita minkäänlaista tiedon tallentamista, koska sen tarkoituksena on olla puhdas ja yksinkertainen toteutus, joka tekee yhden asian hyvin. MaxScalen filtterien perusideologiana on yksinkertaisten filttereiden, jotka tekevät yhden asian, käyttäminen peräkkäin. Tällä säilytetään modulaarisuus ja helppo yksittäisen moduulin konfiguraatio, ja mahdollistetaan tehokkaiden ja kohdistettujen ratkaisujen kehitys. Näistä syistä palomuurimoduuli tuottaa lokiin kirjoitettavaa dataa vain siinä tapauksessa, jos jotain sääntöä rikotaan.

Kun tietokantaan tehtävä operaatio suoritetaan useasta operaatiosta koostuvana transaktiona, palomuurimoduulin ulospäin näkyvä käyttäytyminen ei poikkea suuresti tilanteesta, jolloin käyttäjällä ei ole riittäviä oikeuksia suorittaa kaikkia operaatioita. Asiakasohjelmalle tämä näkyy täysin normaalina tilanteena, eli transaktion sisällä tehtävä operaatio palauttaa virheen puutteellisista oikeuksista tietokantaan. MaxScalen takana oleville tietokantapalvelimille tämä tilanne näkyy vain kesken olevana transaktiona. Transaktioiden peruuttamisen tai hyväksymisen vastuu on aina kyselyn tekijällä. Tämän takia palomuurimoduulin käyttö ei muuta tapaa, jolla transaktioita tehdään tietokantaan.

5 Suorituskykymittausten tarkoitus

Palomuurimoduulin kehityksen yhteydessä kävi ilmi, että johonkin hyvin tarkasti määritettyyn ongelmaan on suhteellisen helppo muodostaa toimiva ratkaisu. Tämä tuo esille suorituskykymittausten yleisimmän ongelman, niiden todellisen tarkoituksen. Jos suorituskykymittauksia ei suunnitella tarkasti, voidaan helposti päätyä tilanteisiin, missä mitattavana ei olekaan mittausten kohde vaan sen suoritusympäristö.

Esimerkki tästä tilanteesta on tietokantavälityspalvelimen suorituskykymittaus, jossa asiakasohjelma sijaitsee erillisessä verkossa, välityspalvelin toisessa ja klusteroitu tietokanta palvelimineen kolmannessa verkossa. Jos kyselytaakka koostuu lyhyistä kyselyistä, joiden suoritus tietokantapalvelimessa kestää suhteellisen lyhyen ajan, johtaa se tilanteeseen, missä suurin osa kyselyyn kuluvasta ajasta on käytetty verkossa liikkumiseen. Tämä johtuu siitä, että asiakasohjelma ei ehdi suorittamaan kyselyitä tarpeeksi nopeasti, jotta tietokantavälityspalvelimen todellista suorituskykyä mitattaisiin, ja mittaustulos mittaakin verkon suorituskykyä. Jos kyselyt taas ovat raskaita eikä palvelin ehdi suorittamaan yhtä kyselyä ennen kuin seuraava odottaa suorittamistaan, asiakasohjelma joutuu odottamaan kyselyiden suoritusten palaamista kyselyiden välissä. Tässä tilanteessa tietokantavälityspalvelimen hyödyt tulevat esille ja mittaustulos mittaa tietokannan suorituskykyä, mutta tässä vaiheessa tietokantapalvelinten määrä ja kyselytaakan sisältö ovat suurimmat tekijät, jotka vaikuttavat mittaustuloksiin. Jos tietokantaklusteri on Multi-Master-tyyppinen ja lukutaakka koostuu suurimmalta osin lukevista kyselyistä, ei sen suorituskyvyn mittaus ole välttämättä niin merkittävä tai toivottu parannus kovinkaan huomattava, jos lukutaakka koostuisikin suurimmalta osin muokkaavista kyselyistä ja taaskaan ei varsinaisesti mitata tietokantavälityspalvelimen suorituskykyä.

Kuten suorituskykyä käsittelevässä osiossa mainittiin, useat tekijät vaikuttavat klusteroidun tietokannan suorituskykyyn, ja luonnollisesti ne kaikki vaikuttavat suorituskykymittauksiin. Kaikkia näitä tekijöitä ei välttämättä voida hallita tai niiden hallitsemiseen vaadittava resurssien määrä voi olla liian suuri. Täten voidaankin kysyä, kuinka hyödyllisiä suorituskykymittaukset ovat?

Ohjelmistokehittäjän näkökulmasta suorituskyvyn mittaaminen on mainio keino saada selville parannukset edellisiin versioihin verrattuna. Jos testejä ajetaan automatisoidusti, voidaan muutokset suorituskyvyssä huomata helposti ja nopeasti.

Yrityksien ja organisaatioiden kannalta suorituskykymittaukset voivat olla keino myydä kehitettyä ohjelmistoa tai todistaa sen toimivuus ja luotettavuus. Loppukäyttäjälle suorituskykymittaukset voivat olla ainoa keino tutkia eri ohjelmistojen välisiä eroja. Ne ovat arvokas työkalu ohjelmiston ostoa tai käyttöä suunnitellessa. Valitettavasti näiden esimerkkinä annettujen ryhmien tavoitteet suorituskykymittauksien suhteen eivät kohtaa. Tämä näkyy varsin hyvin klusteroitujen tietokantojen kanssa. Loppukäyttäjän asemassa ovat yleensä yritykset, ja yritykset ovat myös myyjän asemassa. Koska kaikki tavoittelevat liikevoittoa, muodostuu tilanne, jossa toinen osapuoli haluaisi mahdollisimman kattavat ja tarkat mittaustulokset suorituskyvystä, mieluiten ilmaiseksi, ja toinen osapuoli taas haluaisi saada myydystä tuotteesta mahdollisimman suuren voiton.

Kuten klusteroidun tietokannan suorituskyvyn parantamisen keinoja käsittelevässä osiossa kävi ilmi, ovat suorituskyvyn parannukset kiinni klusterin suoritusympäristöstä. Tämä voidaan myös todeta suorituskykymittausten osalta ja voidaan todeta, että yhtä ja oikeaa suorituskykymittausta ei ole olemassa ja yksi suorituskyvyn mittaus yhdessä tilanteessa voi antaa huomattavasti erilaisia tuloksia toisessa tilanteessa, vaikka käytännössä suorituskyvyssä ei olisikaan eroa. Hyvin tarkkaan määritellyt ja tarkasti hallitut suorituskykymittaukset antavat todellisesti hyödyllistä tietoa, mutta tämän tieto on käytettävää vain hyvin kapealla osa-alueella. Kahden eri organisaation välillä tehtävät suorituskykymittaukset voivat poiketa toisistaan hyvinkin paljon, mikä antaa kuvan huomattavasta erosta suorituskyvyssä. Tämä ero voi kuitenkin johtua yksinkertaisesti suoritusympäristöstä tai tehdystä suorituskykymittauksesta ja käytännössä ero suorituskyvyssä on huomaamaton. Näin voidaankin sanoa suorituskykymittausten olevan täysin päteviä vain niiden omassa suoritusympäristössään.

Tähän perustuukin varsinaisten suorituskykymittausten puute tässä insinöörityössä. Koska palomuurimoduuli kehitettiin ratkaisuksi hyvin tarkasti määriteltyyn ongelmaan, sen suorituskykymittaus olisi näissä tapauksissa suhteellisen turhaa. Toiminnallisuutta mittaavissa testeissä voitiin jo todeta palomuurimoduulin oleva pätevä ratkaisu esimerkkitapauksien ongelmiin. Erilliset suorituskykymittaukset vain toistaisivat toiminnallisuutta mittaavia testejä, ja näiden mittausten tulokset olisivat luotettavia ja käytettäviä vain kyseisen esimerkkitapauksen suoritusympäristössä. Näistä tuloksista ei kuitenkaan olisi mitään käytännön hyötyä, koska palomuurimoduulin perustavana ideologiana oli olla työkalu, jolla voitaisiin vastata johonkin hyvin tarkasti määriteltyyn

kysymykseen. Palomuurimoduulia voisi verrata kirurgin veitseen, joka poistaa vain sen, joka tarvitsee poistamista. Tämän takia kaikki tilanteet, joissa palomuurimoduuli toimii halutulla tavalla, johtavat suorituskykyparannuksiin, jotka olivat jo tiedossa ennen kuin mitään mittauksia on tehty.

6 Yhteenveto

Tämän työn ensimmäiseksi tavoitteiksi oli asetettu klusteroidun tietokannan suorituskyvyn määritelmän selvitys, siihen vaikuttavien tekijöiden tunnistaminen ja parannuskeinojen löytäminen. Työn toisena tavoitteena oli palomuurimoduulin kehitys ja arviointi siitä, kuinka hyvä vastaus se on esimerkkitapausten ongelmiin.

Ensimmäiseksi työssä käytiin läpi klusteroitujen tietokantojen suoritusympäristö ja tutustuttiin siinä käytettäviin teknologioihin. Tässä osiossa saatiin selville replikaation tarkoitus ja kuinka se käytännössä toimii, ja minkälaisia toteutuksia replikaation eri muodoista on.

Tämän jälkeen siirryttiin klusteroidun tietokannan suorituskyvyn määritelmään ja siihen vaikuttaviin tekijöihin. Näiden kautta tuotiin esille erilaisia keinoja parantaa klusteroidun tietokannan suorituskykyä.

Työn käytännön osuudessa kehitettiin MaxScale-ohjelmistoon palomuurimoduuli. Tätä moduulia varten oli kerätty kolme esimerkkitapausta, jotka kuvasivat kolmea erilaista ongelmatilannetta. Näitä esimerkkitapauksia apuna käyttäen käytiin läpi moduulin tavoitteet, sisäinen toiminta ja logiikka, jolla kyselyjä estetään. Viimeisenä osana käytännön osuutta tehtiin toimivuuden testaus, jossa moduuli todettiin toimivaksi, ja tutkittiin, miten moduuli näkyy loppukäyttäjille ja ylläpitäjille.

Lopuksi pohdittiin suorituskyvyn mittauksen tarkoitusta, milloin ne ovat todella hyödyllisiä ja miksi niitä ei aina voi vertailla toisiinsa.

Työn ensimmäinen tavoite saavutettiin, kun määriteltiin suorituskyky monimuotoiseksi asiaksi, jonka yleinen analysointi ei välttämättä tuota toivottuja tuloksia. Osana tätä tavoitetta oli myös suorituskykyyn vaikuttavien tekijöiden tunnistaminen. Varsinkin kyselytaakan tasapaino muokkaavien ja lukevien kyselyiden välillä,

tietokantapalvelinten määrä ja niiden fyysinen sijainti ja replikaatiossa käytettävä teknologia tunnistettiin tärkeiksi tekijöiksi suorituskyvyn kannalta. Myös suorituskyvyn parannuskeinojen tutkiminen oli osana ensimmäistä tavoitetta. Tämäkin osa ensimmäistä tavoitetta saavutettiin, kun arvioitiin mahdollisia suorituskyvyn parannuskeinoja. Näistä parannuskeinoista toteutettiin yksi käytännön esimerkin muodossa, jonka tekeminen ja analysoiminen olivat insinöörityön toinen tavoite.

Työn toinen tavoite, palomuurimoduulin kehitys, tuki työn ensimmäistä tavoitetta ja antoi käytännön esimerkin keinosta parantaa tietokantaklusterin suorituskykyä. Kehitetty palomuurimoduuli todettiin toimivaksi ratkaisuksi, kun toimintaympäristö oli tarkkaan määritelty ja suorituskykyongelmien syyt tiedettiin suhteellisen tarkasti. Tämä toi esiin suorituskyvyn mittauksen ongelmallisuuden, jonka merkityksiin syvennyttiin aihealuetta analysoivassa ja pohtivassa luvussa.

Näiden tehtyjen asioiden perusteella asetetut tavoitteet saavutettiin. Moduuli voidaan todeta toimivaksi ratkaisuksi esimerkkitapausten ongelmiin ja tästä voidaan tehdä johtopäätös, että ongelmatekijöiden eliminoiminen parantaa klusteroidun tietokannan suorituskykyä.

7 Lähdeluettelo

[1] Ilari Moilanen, Tietokantaklusterit, Marraskuu 28, 2007.

[2] Cory Janssen. (2014, Lokakuu) Technopedia. [Verkkolähde]. http://www.techopedia.com/definition/17/clustering-databases. Luettu 13 Lokakuu 2014.

[3] Janukowicz Reinsel, Datacenter SSDs : Sol id Foot i n g f o r G rowt h, Tammikuu 2008, Sponsored by Samsung.

[4] Free Software Foundation, Inc. (2014, Marraskuu) GDB: The GNU Project Debugger. [Verkkolähde]. http://www.gnu.org/software/gdb/. Luettu 26 Marraskuu 2014.

[5] Database cluster and load balancing, Lokakuu 13, 2014.

[6] Michael Stonebraker and Rick Cattell, "Ten Rules for Scalable Performance in “Simple Operation”," *Communications of the ACM*, pp. 72-80, 2011.

[7] (2014, Kesäkuu) PostgreSQL Wiki. [Verkkolähde]. https://wiki.postgresql.org/wiki/Logical_Log_Streaming_Replication. Luettu 12 Joulukuu 2014.

[8] (2014, Joulukuu) Wikipedia.org. [Verkkolähde]. http://en.wikipedia.org/wiki/Mainframe_computer. Luettu 21 Joulukuu 2014.

[9] Ph.D. Thomas E. Beach. (2012, Tammikuu) Computer Concepts and Terminology. [Verkkolähde]. http://www.unm.edu/~tbeach/terms/types.html. Luettu 21 Joulukuu 2014.

[10] Raatikka Wolski. (2006, Toukokuu) Performance Measurement and Tuning. [Verkkolähde]. http://antoni.wolski.fi/sol-pub/awo-vraa-isas06.pdf. Luettu 16 Lokakuu 2014.

[11] cplusplus.com. (2014) cplusplus.com. [Verkkolähde]. http://www.cplusplus.com/reference/regex/. Luettu 23 Joulukuu 2014.

[12] IBM. (2014, Marraskuu) IBM. [Verkkolähde]. http://www.ibm.com/big-data/us/en/. Luettu 12 Marraskuu 2014.

[13] Oracle. (2014, Lokakuu) MySQL. [Verkkolähde]. http://dev.mysql.com/doc/refman/5.0/en/replication.html. Luettu 3 Lokakuu 2014.

[14] Oracle. (2014, Lokakuu) Content Storage in the Media & Entertainment Industry. [Verkkolähde]. http://www.oracle.com/us/industries/media-entertainment/content-storage-br-393458.pdf. Luettu 16 Lokakuu 2014.

[15] Wikimedia Foundation Inc. (2014, Marraskuu) Jokerimerkki. [Verkkolähde]. http://fi.wikipedia.org/wiki/Jokerimerkki. Luettu 26 Marraskuu 2014.